A Heterogeneous Computing Framework for Computational Finance

Gordon Inggs, David Thomas and Wayne Luk

Imperial College London

- Introduction
- Computational Finance
- Forward Financial Framework (F^3)
- Experimentation
- Concluding Remarks

Introduction

- Financial Derivatives are a critical component of modern commerce
- Computationally-intensive pricing models are used to value derivative products
- The Financial Industry is a major consumer of high-end computing

Use Case

- Large Institution
- Many Users
- Many Computational Tasks (with dependencies)
- Heterogeneous Computing Resources

Our Work

- Domain Specific Approach

- Three Challenges:
  1) Implementation → Efficient, Automated across range of platforms
  2) Coping with Dependencies → Removal of redundant computations
  3) Partitioning → Domain Knowledge-guided Partitioning

Computational Finance

- Application Background:
  - Forward Looking Derivatives
  - Derivative Valuation
  - Monte Carlo Algorithm

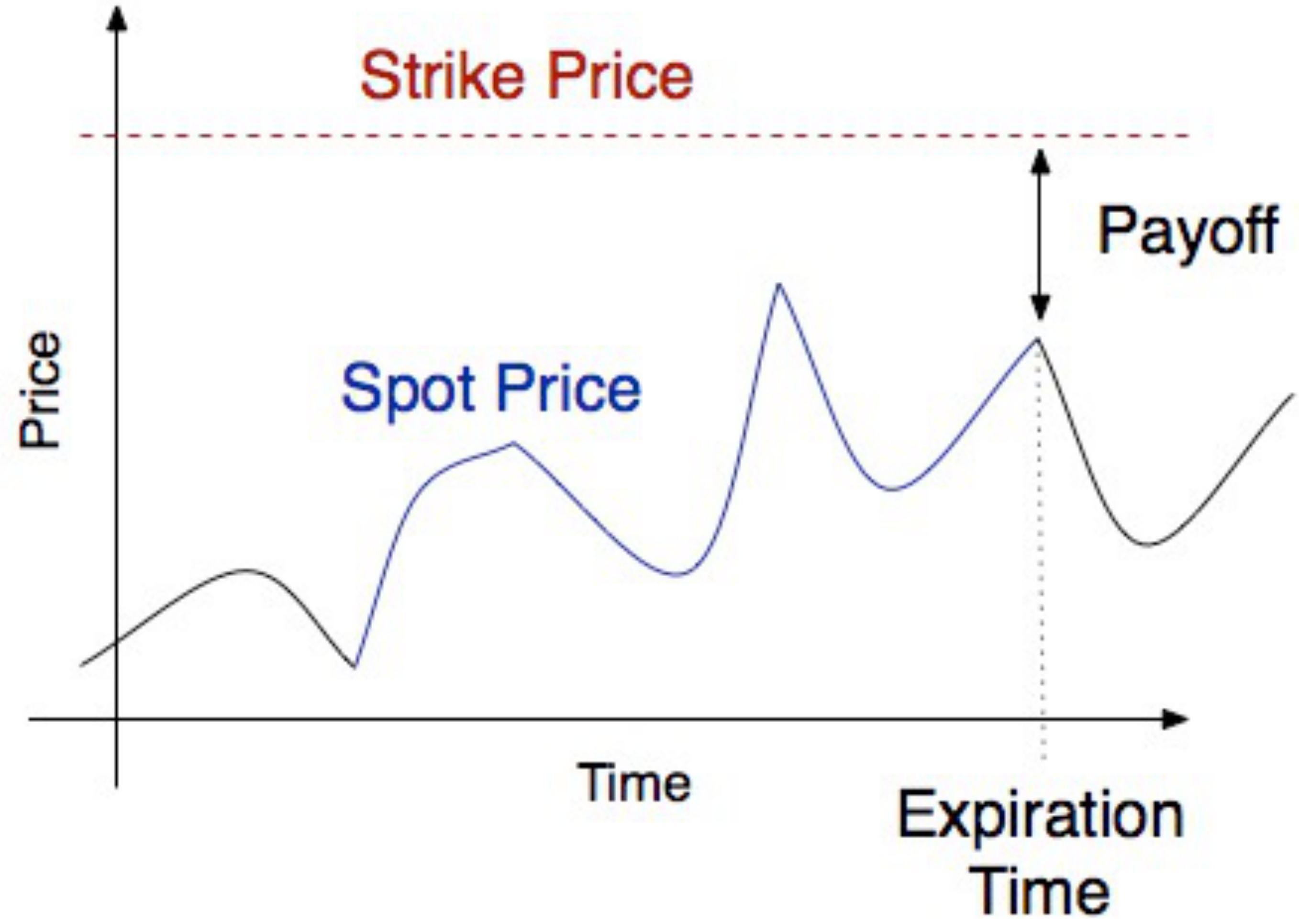
- Computational Domain:
  - Fundamental Concepts
  - Domain Relationships

Forward Looking Derivatives

- An option contract grants the <u>right</u> to buy or sell a defined asset at a defined point in the future, for a defined price

- A <u>forward-looking option</u> is one with a single defined point at which it may be exercised

How to value a Derivative



Monte Carlo Algorithm

- A popular method – flexible, robust
- Use a model of the asset to simulate the underlying asset through its life-time:

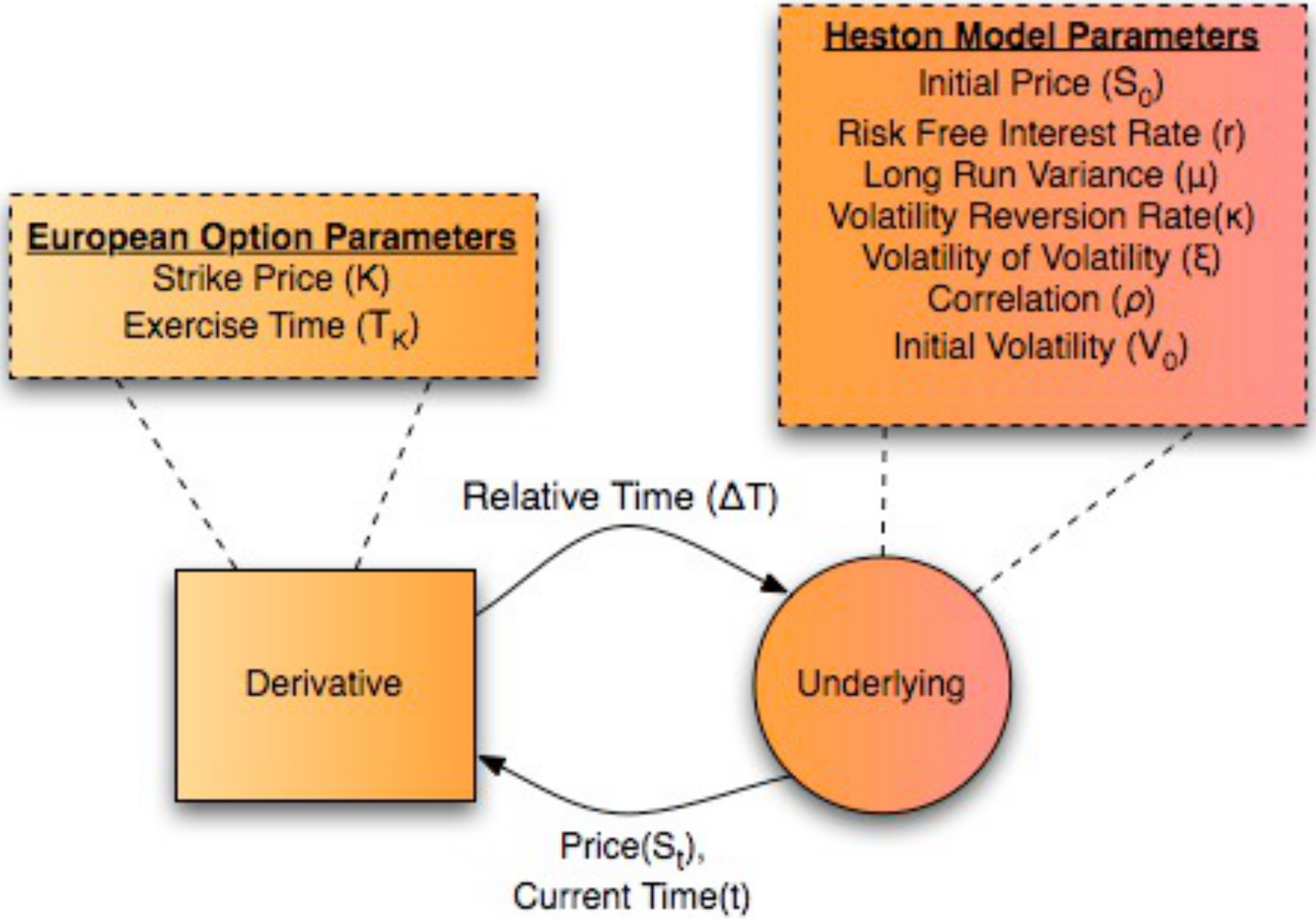
$$dS = \mu\, dt + \sigma\, S\, dW_t$$

- Average the payoff over many sample paths:

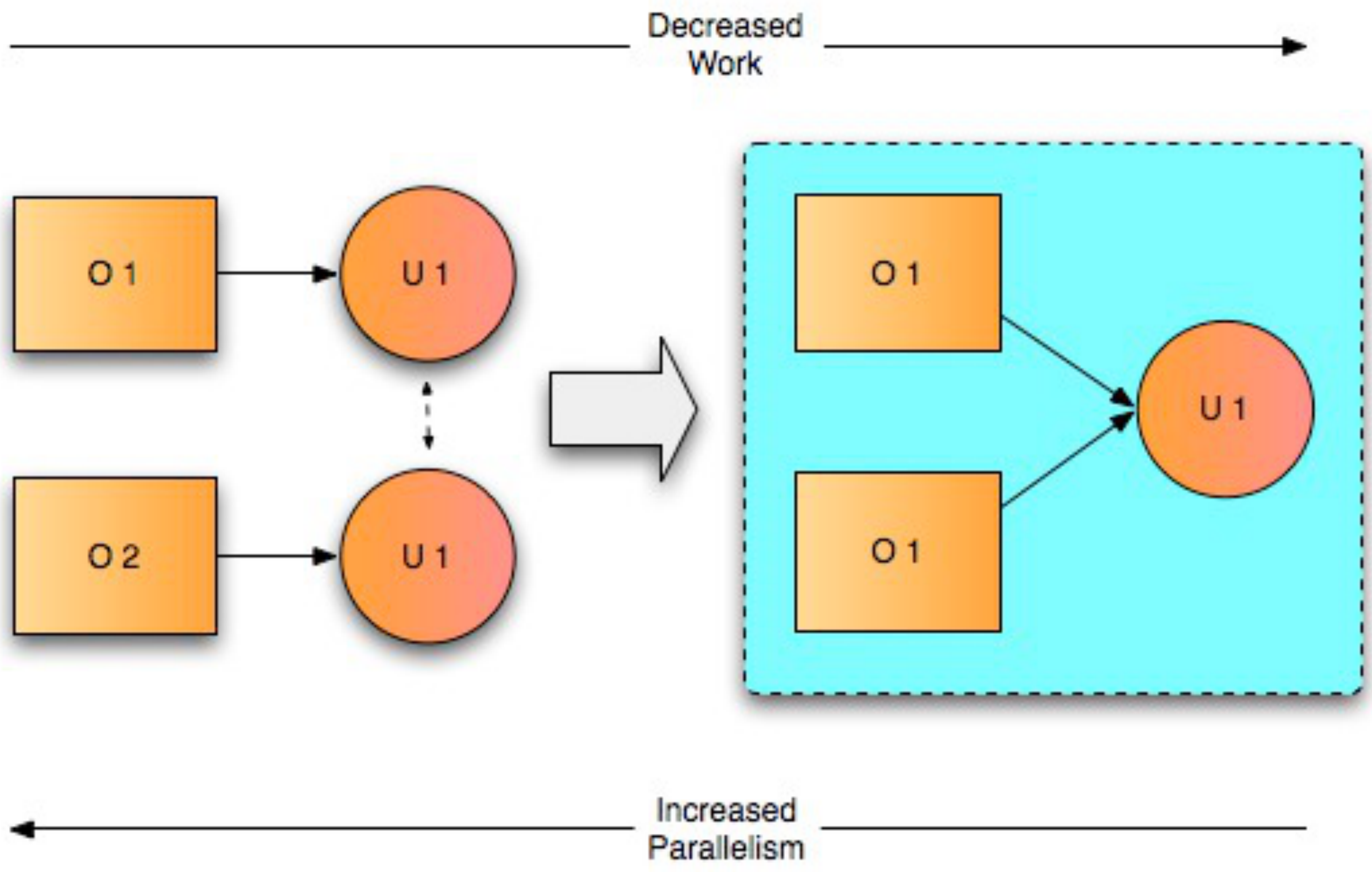
$$V_t \approx max\left(e^{-r(T-t)} \frac{1}{N} \sum_{k=0}^{N-1} S_{k,T} - K, 0\right)$$

Computational Finance Domain Concepts

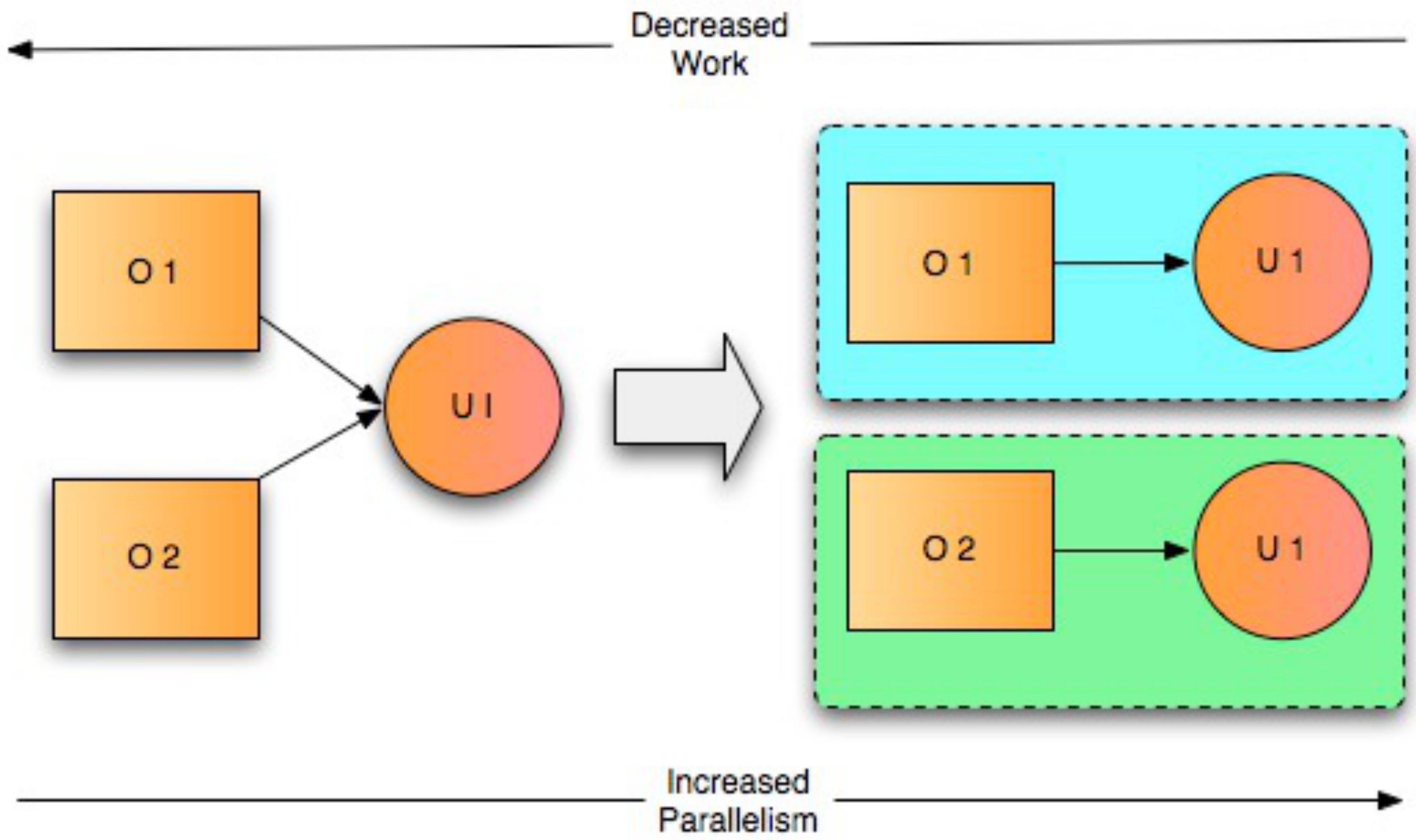
Computational Finance Domain Concepts



Domain Dependencies - "Fusion"



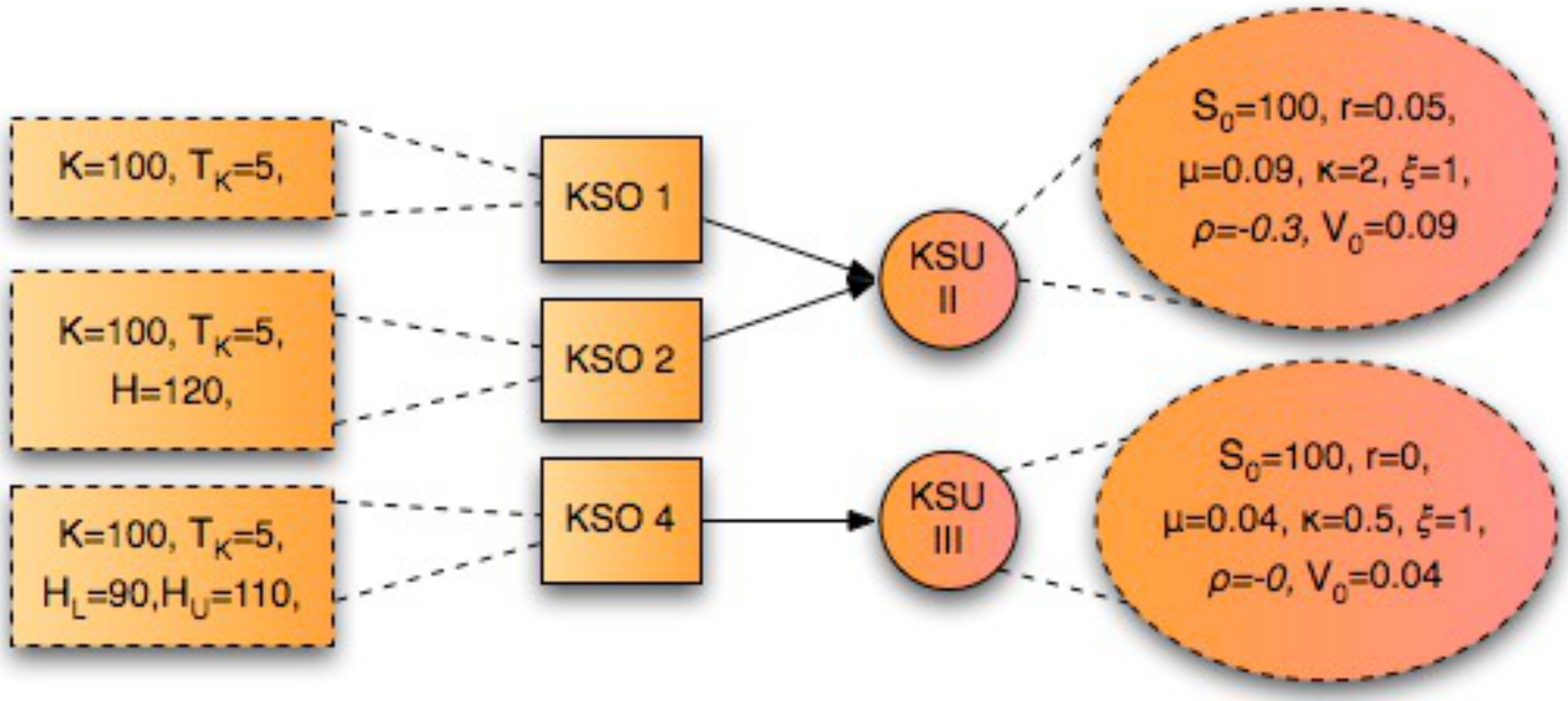
Domain Dependencies - “Fission”



Forward Financial Framework (F^3)

- <u>Application Framework</u> vs (Domain Specific Language)
- Supports range of option types, underlying models and Monte Carlo Pricing
- Multicore CPUs, GPUs (via OpenCL) and Maxeler FPGA
- Open Source

F^3 Fundamentals

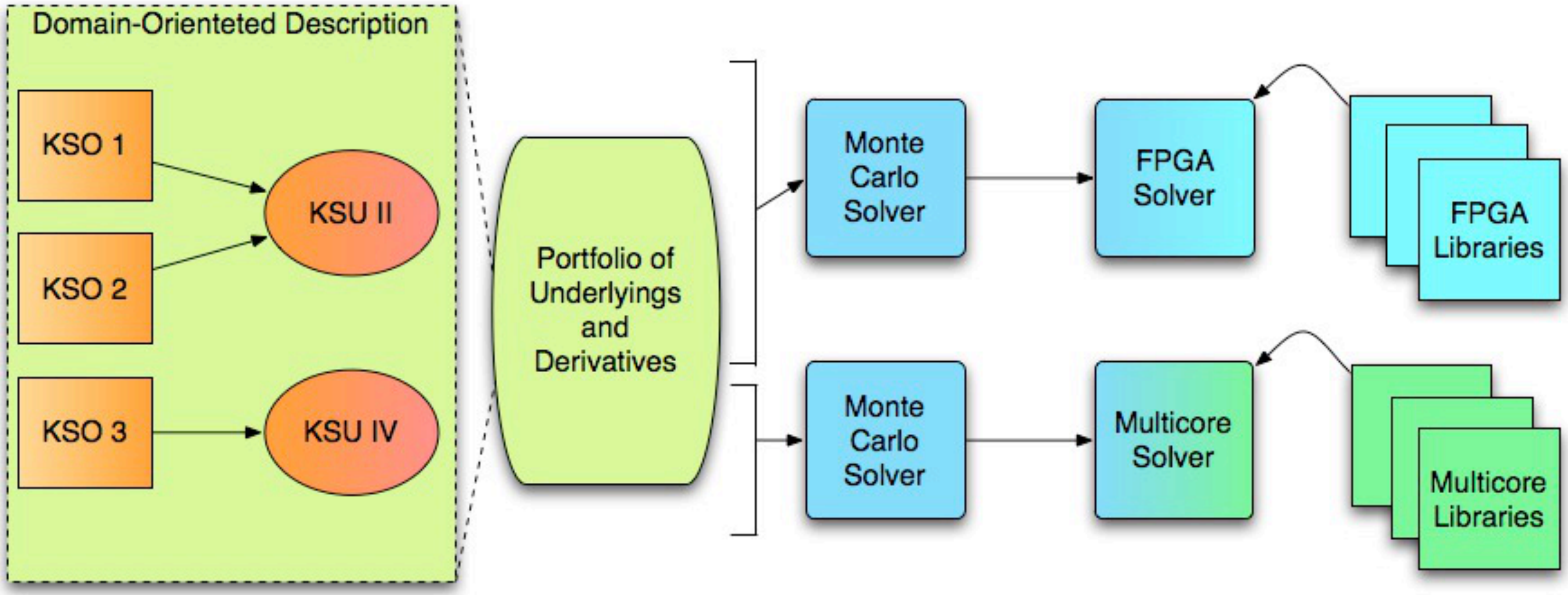


F^3 Fundamentals (II)

```
#Declaring the Underlyings
Heston_II = Heston_Underlying(0.05,100,0.09,1,-0.3,2,0.09)
Heston_IV = Heston_Underlying(0,100,0.09,1,-0.3,1,0.09)

#Declaring the Options
Option_1 = European_Option (Heston_II,True,Current_Price,5)
Option_2 = Barrier_Option (Heston_II,True,Current_Price,5,4096,True,120)
Option_3 = Barrier_Option (Heston_IV,True,Current_Price,5,4096,True,120)
```

F^3 Implementation Flow



F^3 Implementation Flow (II)

```
#Creating the platforms and solver objects
multicore_cpu = Multicore_CPU()
maxeler_fpga = Maxeler_FPGA()
mc_solver_cpu = MonteCarlo(derivative=[Option_1,Option_2],10e7,
    multicore_cpu)
mc_solver_fpga = MonteCarlo(derivative=[Option_3],10e7,
    maxeler_fpga)

#Calling the CPU solver to generate, compile and execute
mc_solver_cpu.generate()
mc_solver_cpu.compile()
results = mc_solver_cpu.execute()
```

F^3 incorporates Domain Knowledge

- Structure → The framework's objects mirrors the domain concepts
- General Optimisation → The “fusion” rule is used to avoid redundant computations
- Partitioning → The “fission” rule allows for flexibility during partitioning

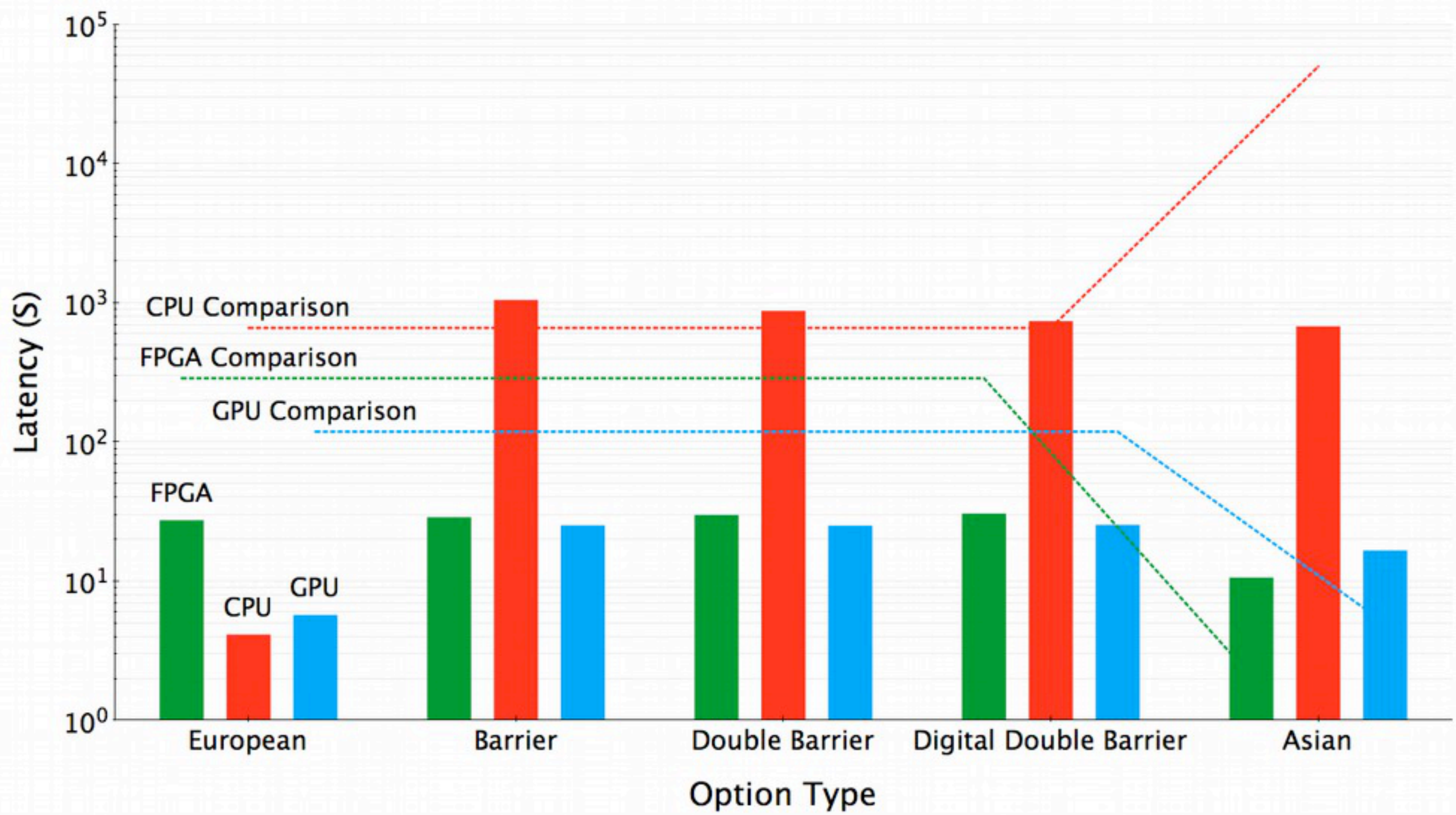
Experimentation

- 3 Claims:
  1) Efficient, Automated Implementations
  2) Removal of Redundant Computation
  3) Domain Knowledge can guide partitioning

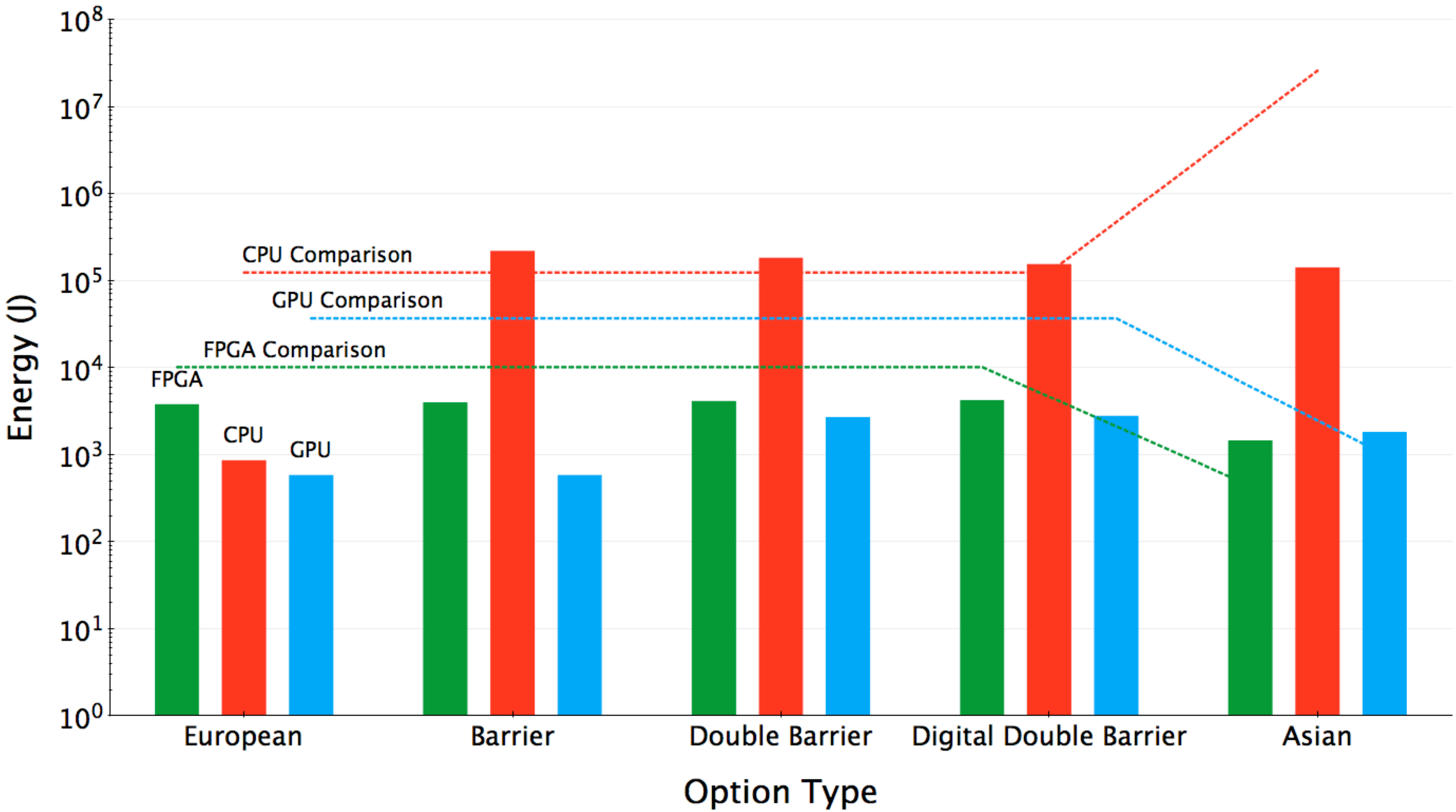
1^{st} Claim – Efficient, Automated Implementations

- Compare against external implementations:
  - Kaiserslautern Barrier Option Pricing Benchmark
  - Imperial College Asian Option Pricing
- 10 Million Simulations, 4096 Points per Sim
- Multicore CPU (8 core Intel Core i7), GPU* (AMD FirePro W5000) and FPGA (Maxeler Max3 – Virtex 6)
- Latency and Energy

Implementation Evaluation (Latency)



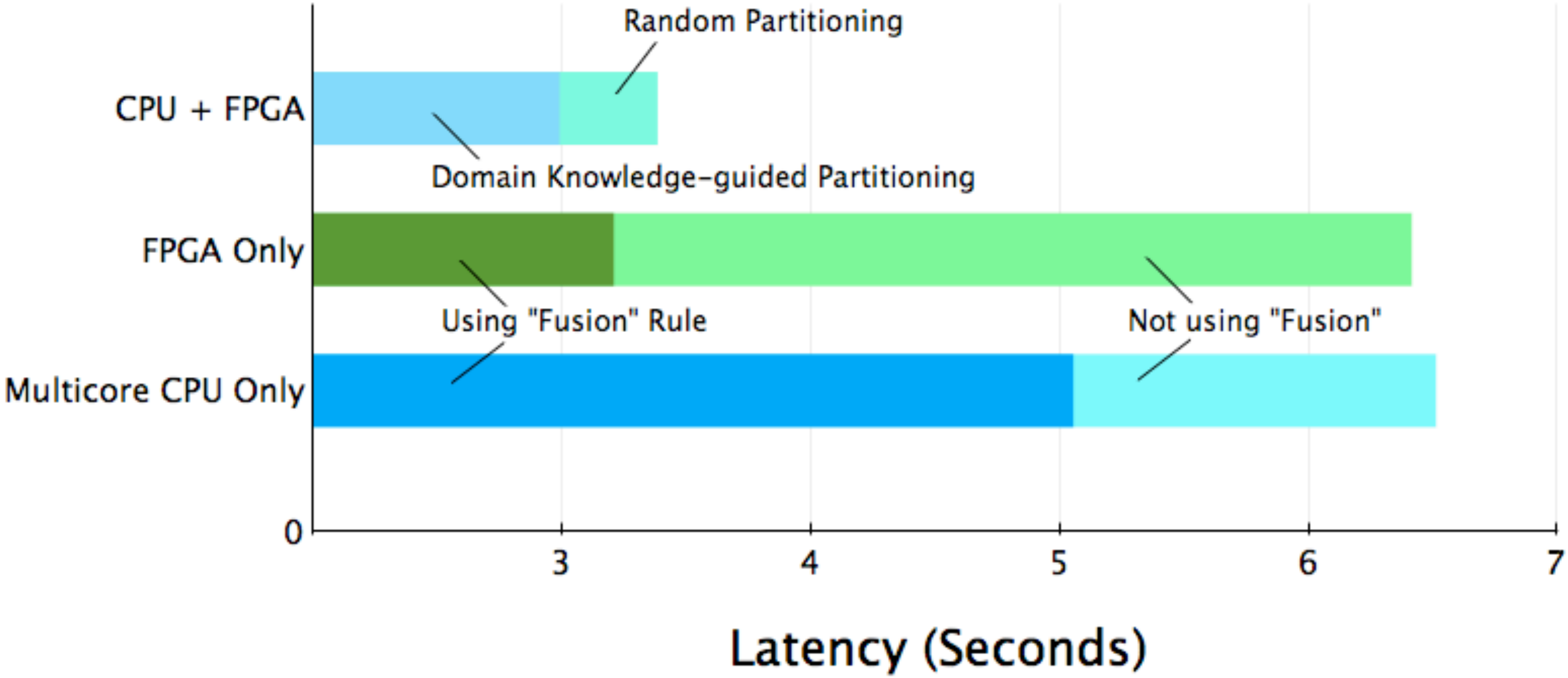
Implementation Evaluation (Energy)



2^{nd} and 3^{rd} Claims – Exploiting Domain Knowledge

- Construct Portfolio out of Kaiserslatuarn Benchmark + Imperial Asian Option
- Multicore CPU and FPGA, independently and together
- Latency

Domain Rule Optimisations



Conclusion

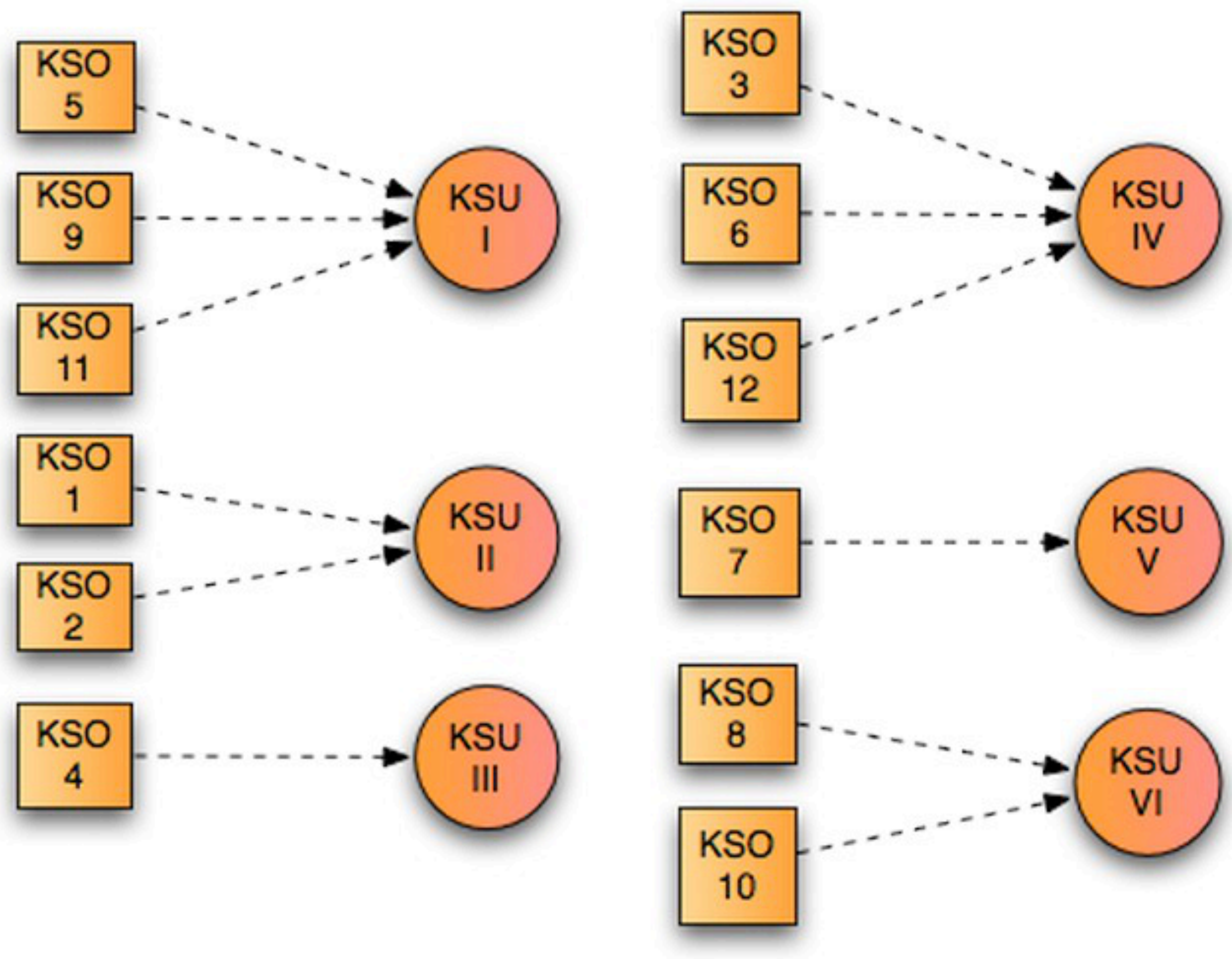
- Our approach to Computational Finance as a Problem Domain
- Contributions:
  1) Efficient, Automated Implementations
  2) Removal of Redundant Computations
  3) Domain-Knowledge Guided Partitioning

- Is Domain Specificity worth it?

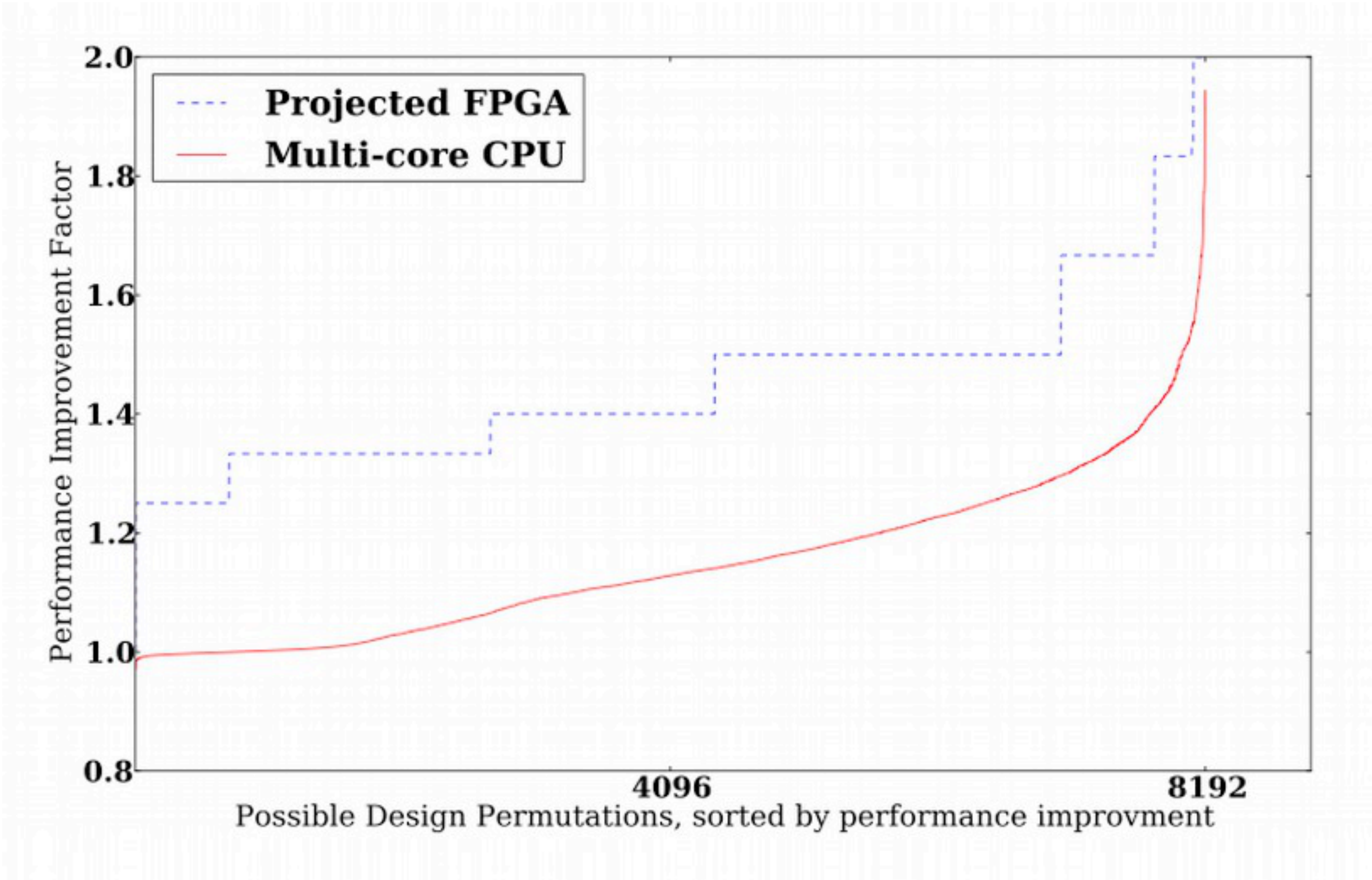
Future Work

- More Heterogeneity!
- Runtime Characterisation and Modelling
- End-User Exploration of the Design Space

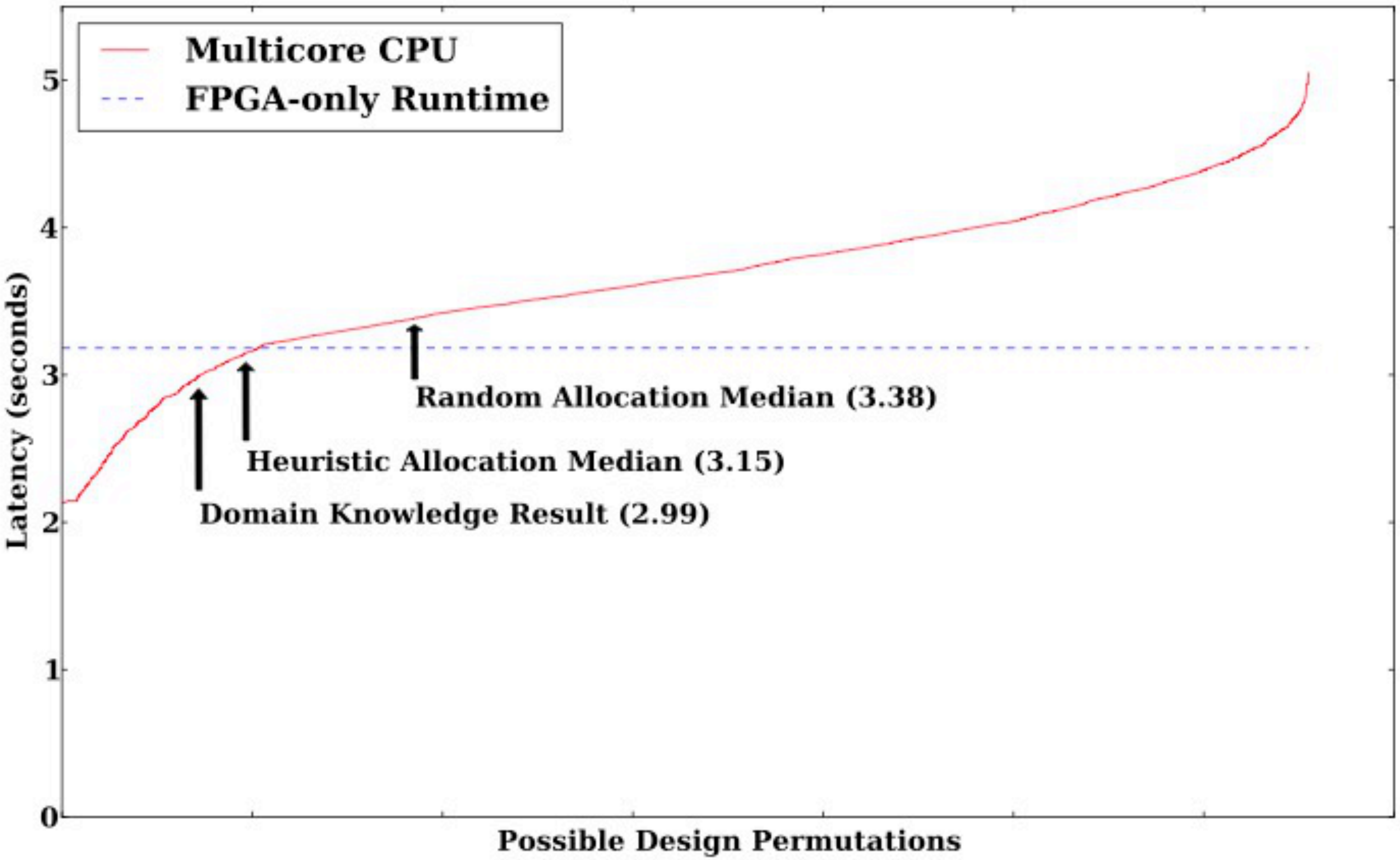
Kaiserslatuarn Benchmark



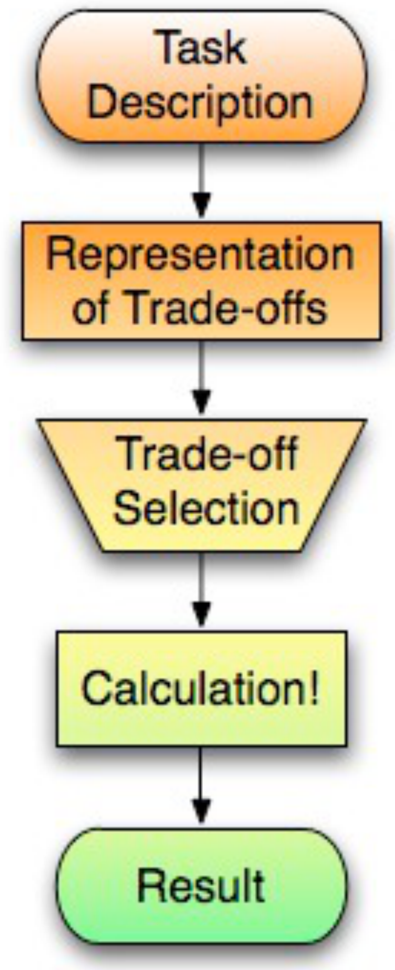
Removing Redundant Computations



DK Guided Partitioning



Future Work



Task Description
Representation of Trade-offs
Trade-off Selection
Calculation!
Result